# Univariate polynomial factorization over large finite fields

Joris van der Hoeven, Grégoire Lecerf

▶ **To cite this version:**

Joris van der Hoeven, Grégoire Lecerf. Univariate polynomial factorization over large finite fields. Applicable Algebra in Engineering, Communication and Computing, 2022, 10.1007/s00200-021-00536-1 . hal-03019847

**HAL Id: hal-03019847**
**https://hal.science/hal-03019847v1**

Submitted on 23 Nov 2020

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Univariate polynomial factorization over large finite fields*†

JORIS VAN DER HOEVEN[ab], GRÉGOIRE LECERF[ac]

*a*. CNRS, École polytechnique, Institut Polytechnique de Paris
Laboratoire d'informatique de l'École polytechnique (LIX, UMR 7161)
1, rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing, CS35003
91120 Palaiseau, France

*b*. *Email:* vdhoeven@lix.polytechnique.fr
*c*. *Email:* lecerf@lix.polytechnique.fr

*Preliminary version of November 23, 2020*

The best known asymptotic bit complexity bound for factoring univariate polynomials over finite fields grows with the input degree to a power close to 1.5, and with the square of the bitsize of the ground field. It relies on a variant of the Cantor–Zassenhaus algorithm which exploits fast modular composition. Using techniques by Kaltofen and Shoup, we prove a refinement of this bound when the finite field has a large extension degree over its prime field. We also present fast practical algorithms for the case when the extension degree is smooth.

## 1. INTRODUCTION

The usual *primitive element representation* of the finite field $\mathbb{F}_q$ with $q = p^\kappa$ elements is $\mathbb{F}_p[z]/(\theta(z))$ with $p$ prime and $\theta \in \mathbb{F}_p[z]$ irreducible and monic of degree $\kappa \geqslant 1$. For this representation, von zur Gathen, Kaltofen, and Shoup proposed several efficient algorithms for the irreducible factorization of a polynomial $f$ of degree $d$ in $\mathbb{F}_q[x]$. One of them is a variant of the Cantor–Zassenhaus method for which large powers of polynomials modulo $f$ are computed using modular composition [24, section 2]. Now Kedlaya and Umans designed a theoretically efficient algorithm for modular composition [25, 26]. Consequently, using a probabilistic algorithm of Las Vegas type, the polynomial $f$ can be factored in expected time

$$d^{1.5+o(1)} \log^{1+o(1)} q + \tilde{O}(d \log^2 q). \tag{1.1}$$

It turns out that the second term of (1.1) is suboptimal when $\log q$ becomes large. The purpose of the present paper is to prove the expected complexity bound

$$(d^{0.5+\epsilon(d)} + \kappa^{\epsilon(\kappa)} + \log p)\, \tilde{O}(d \log q),$$

*. This paper is part of a project that has received funding from the French "Agence de l'innovation de défense".

†. This article has been written using GNU TEXMACS [15].

where $\epsilon(d) = O\Big(\Big(\frac{\log\log d}{\log d}\Big)^{1/2}\Big)$; see Corollary 5.1. For this, we rely on ideas by Kaltofen and Shoup from [23]. In addition, we present improved complexity bounds for the case when the extension degree $\kappa$ is smooth. These bounds rely on practically efficient algorithms for modular composition that were designed for this case in [16].

## 1.1. Notations

Given a commutative ring $\mathbb{A}$ and $d \in \mathbb{N}$, let $\mathbb{A}[x]_{<d} := \{f \in \mathbb{A}[x] : \deg f < d\}$. Given $x \in \mathbb{R}$, we define $\lfloor x \rfloor := \max\{n \in \mathbb{Z} : n \leqslant x\}$ and $\lceil x \rceil := \min\{n \in \mathbb{Z} : n \geqslant x\}$. For any positive integer $n$, we set

$$\lg n := \lfloor \log_2 n \rfloor,$$

so that $n/2 < 2^{\lfloor \log_2 n \rfloor} \leqslant n$. We will freely use the *soft-Oh* notation: $f(n) = \tilde{O}(g(n))$ means that $f(n) = g(n)\,(\log(g(n)))^{O(1)}$, as in [9].

Until section 5.2, we assume the standard complexity model of a Turing machine with a sufficiently large number of tapes. For probabilistic algorithms, the Turing machine is assumed to provide an instruction for writing a "random bit" to one of the tapes. This instruction takes a constant time. All probabilistic algorithms in this paper are of Las Vegas type; this guarantees that all computed results are correct, but the execution time is a random variable.

Until section 5 we consider *abstract finite fields* $\mathbb{F}_q$, whose internal representations are not necessarily prescribed, and we rely on the following assumptions and notations:

- Additions and subtractions in $\mathbb{F}_q$ take linear time.
- $\mathsf{M}_{\mathbb{F}_q}(d)$ denotes an upper bound for the bit complexity of polynomial products in degree $<d$ over $\mathbb{F}_q$. It is convenient to make the regularity assumptions that $\mathsf{M}_{\mathbb{F}_q}(d)/d$ is nondecreasing as a function of $d$ and that $\mathsf{M}_{\mathbb{F}_q}(nd) = O(n\,\mathsf{M}_{\mathbb{F}_q}(d))$ for $n = O(d)$.
- $\mathsf{D}_{\mathbb{F}_q}$ upper bounds the time needed to invert one element in $\mathbb{F}_q$.
- $\mathsf{C}_{\mathbb{F}_q}(d)$ is an upper bound for the time to compute $f(x) \circ g(x) \bmod h(x)$ for $f, g \in \mathbb{F}_q[x]_{<d}$ and monic $h \in \mathbb{F}_q[x]$ of degree $d$.
- $\mathsf{C}_{\mathbb{F}_q}(d; N)$ is an upper bound for the time to compute $f_1(x) \circ g(x), \ldots, f_N(x) \circ g(x)$ modulo $h(x)$ for $f_1, \ldots, f_N, g \in \mathbb{F}_q[x]_{<d}$ and monic $h \in \mathbb{F}_q[x]$ of degree $d$.
- $\Phi_{\mathbb{F}_q}$ is an upper bound for the time to compute $a^{p^e}$, given $a \in \mathbb{F}_q$ and $e \in \{1, 2, 4, \ldots, 2^{\lg \kappa}\}$.

## 1.2. Related work

For general algorithms for finite fields, we recommend the textbooks [4, 9, 28, 35] and more specifically [9, chapter 14], as well as [8, 21] for historical references. In this paper we adopt the asymptotic complexity point of view, while allowing for randomized algorithms of Las Vegas type.

Early theoretical and practical complexity bounds for factorizing polynomials over finite fields go back to the sixties [2, 3]. In the eighties, Cantor and Zassenhaus [5] popularized distinct-degree and equal-degree factorizations. Improved complexity bounds and fast implementations were explored by Shoup [34]. He and von zur Gathen introduced the iterated Frobenius technique and the "baby-step giant-step" for the distinct degree factorization [11]. They reached softly quadratic time in the degree via fast multipoint evaluation. Together with Kaltofen, they also showed how to exploit modular composition [22, 24] and proved sub-quadratic complexity bounds.

In 2008, Kedlaya and Umans showed that the complexity exponent of modular composition over finite fields is arbitrarily close to one [25, 26]. As a corollary, the complexity exponent of polynomial factorization in the degree is arbitrarily close to 1.5.

Von zur Gathen and Seroussi [10] showed that factoring quadratic polynomials with arithmetic circuits or straight-line programs over $\mathbb{F}_q$ requires $\Omega(\log q)$ operations in $\mathbb{F}_q$. However, this does not provide a lower bound $\Omega(\log^2 q)$ for boolean arithmetic circuits. In fact, subquadratic upper bounds indeed exist for the stronger model of arithmetic circuits over the prime subfield $\mathbb{F}_p$ of $\mathbb{F}_q$: the combination of [23, Theorem 3] and fast modular composition from [19] allows for degree $d$ factorization in $\mathbb{F}_q[x]$ in expected time

$$(d + \kappa^{\epsilon(\kappa)} + \log p)\, \tilde{O}(d\kappa \log p).$$

This bound is indeed subquadratic and even quasi-optimal in $\kappa$. In this paper, we also achieve a subquadratic dependence on $d$.

In order to save modular compositions, Rabin introduced a randomization process that uses random shifts of the variable [32]. This turns out to be useful in practice, especially when the ground field $\mathbb{F}_q$ is sufficiently large. We briefly revisit this strategy, following Ben-Or [1] and [9, Exercise 14.17].

Unfortunately, Kedlaya and Umans' fast algorithm for modular composition has not yet given rise to fast practical implementations. In [16], we have developed alternative algorithms for modular composition which are of practical interest when the extension degree $\kappa$ of $\mathbb{F}_q$ over $\mathbb{F}_p$ is composite or smooth. In section 6, we study the application of these algorithms to polynomial factorization.

The probabilistic arguments that are used to derive the above complexity bounds become easier when ignoring all hidden constants in the "$O$". Sharper bounds can be obtained by refining the probability analyses; we refer to [6, 7] for details.

## 1.3. Contributions

In this paper we heavily rely on known techniques and results on polynomial factorization (by von zur Gathen, Kaltofen, Shoup, among others) and modular composition (by Kedlaya, Umans, among others). Through a new combination of these results, our main aim is to sharpen the complexity bounds for polynomial factorization and irreducibility testing over finite fields. The improved bounds are most interesting when factoring over a field $\mathbb{F}_q$ with a large and smooth extension degree $\kappa$ over its prime field $\mathbb{F}_p$.

Let us briefly mention a few technical novelties. Our finite field framework is designed to support a fine grained complexity analysis for specific modular composition algorithms, for sharing such compositions, and for computing factors up to a given degree. We explicitly show how complexities depend on the degrees of the irreducible factors to be computed.

Compared to the algorithm of Kaltofen and Shoup [23], our Algorithm 4.2 for equal-degree factorization successively computes pseudo-traces over $\mathbb{F}_q$ and then over $\mathbb{F}_p$. The computation of Frobenius maps is accelerated through improved caching. Our approach also combines a bit better with Rabin's randomization; see section 4.5.

We further indicate opportunities to exploit shared arguments between several modular compositions, by expressing our complexity bounds in terms of $\mathsf{C}_{\mathbb{F}_q}(d; N)$ instead of $\mathsf{C}_{\mathbb{F}_q}(d)$, when possible. We clearly have $\mathsf{C}_{\mathbb{F}_q}(d; N) \leqslant N\, \mathsf{C}_{\mathbb{F}_q}(d)$, but better bounds might be achievable through precomputations based on the shared arguments. We refer to [18, 29] for partial evidence in this direction under suitable genericity assumptions.

Our main complexity bounds are stated in section 4.4. The remainder of the paper is devoted to corollaries of these bounds for special cases. In section 5, we start with some theoretical consequences that rely on the Kedlaya–Umans algorithm for modular composition [26] and variants from [19]. We both consider the case when $\mathbb{F}_q$ is presented as a primitive extension of $\mathbb{F}_p$ and the case when $\mathbb{F}_q = \mathbb{K}_t$, where $\mathbb{F}_p = \mathbb{K}_0 \subset \cdots \subset \mathbb{K}_t = \mathbb{F}_q$ is a "triangular tower".

At the time being, it seems unlikely that Kedlaya and Umans' algorithm can be implemented in a way that makes it efficient for practical purposes; see [19, Conclusion]. In our final section, we consider a more practical alternative approach to modular composition [16], which requires the extension degree $\kappa$ to be composite, and which is most efficient when $\kappa$ is smooth. We present new complexity bounds for this case, which we expect to be of practical interest when $\kappa$ becomes large.

## 2. PSEUDO-FROBENIUS MAPS

The central ingredient to fast polynomial factorization is the efficient evaluation of Frobenius maps. Besides the absolute Frobenius map $\mathbb{F}_q \to \mathbb{F}_q; a \mapsto a^p$ of $\mathbb{F}_q$ over its prime field $\mathbb{F}_p$, we will consider pseudo-Frobenius maps. Consider an extension $\mathbb{A} := \mathbb{F}_q[x]/(f(x))$, where $f \in \mathbb{F}_q[x]$ is monic of degree $d$ and not necessarily irreducible. The $\mathbb{F}_q$-linear map $a \mapsto a^q$ is called the *pseudo-Frobenius map* of $\mathbb{A}$. The map $a \mapsto a^p$ is called the *absolute pseudo-Frobenius map* of $\mathbb{A}$; it is only $\mathbb{F}_p$-linear.

### 2.1. Absolute Frobenius maps

Recall that $q = p^\kappa$ and consider a primitive representation $\mathbb{F}_q \equiv \mathbb{F}_p[z]/(\theta(z))$ for $\mathbb{F}_q$. Let us show how to evaluate iterated Frobenius maps $a \mapsto a^{p^e}$ of $\mathbb{F}_q$ using modular composition. We introduce the auxiliary sequence

$$\mathcal{E}_{\mathbb{F}_q} := (\mathcal{E}_i(z))_{0 \leqslant i \leqslant \lg \kappa}$$

where

$$\mathcal{E}_i(z) \;:=\; z^{p^{2^i}} \bmod \theta(z), \qquad i = 0, \ldots, \lg \kappa. \tag{2.1}$$

The sequence $\mathcal{E}_{\mathbb{F}_q}$ enables us to efficiently compute $p^{2^i}$-th powers and then general $p^e$-th powers, as follows:

LEMMA 2.1. *Let $\mathcal{E}_{\mathbb{F}_q}$ be given. For all $a(z) \in \mathbb{F}_p[z]_{<\kappa}$ and $0 \leqslant i \leqslant \lg \kappa$, we have*

$$a(z)^{p^{2^i}} \bmod \theta(z) \;=\; a(z) \circ \mathcal{E}_i(z) \bmod \theta(z).$$

*In particular,*

$$\Phi_{\mathbb{F}_q} \;=\; \mathsf{C}_{\mathbb{F}_p}(\kappa) + O(\log q).$$

**Proof.** We verify that

$$a(z) \circ \mathcal{E}_i(z) \bmod \theta(z) = a(z) \circ z^{p^{2^i}} \bmod \theta(z)$$

and that

$$a(z) \circ z^{p^{2^i}} = a(z^{p^{2^i}}) = a(z)^{p^{2^i}}.$$

The cost analysis is straightforward from the definitions. □

The cost function $\Phi_{\mathbb{F}_q}$ corresponds to the time needed to iterate the absolute Frobenius map $a \mapsto a^p$ a number of times that is a power of two. For an arbitrary number of iterations we may use the following general lemma.

LEMMA 2.2. *For all $a \in \mathbb{F}_q$ and $e \in \{0, \ldots, \kappa-1\}$ with binary expansion $e = 2^{i_1} + \cdots + 2^{i_l}$ with $i_1 > \cdots > i_l$, we have*

$$a^{p^e} = (((a^{p^{2^{i_1}}})^{p^{2^{i_2}}})^{\cdots})^{p^{2^{i_l}}}.$$

*In particular, computing $a^{p^e}$ takes $O(\Phi_{\mathbb{F}_q} \log \kappa)$ operations.*

**Proof.** The proof is straightforward since $l = O(\log \kappa)$. □

The computation of $\mathcal{E}_{\mathbb{F}_q}$ can be done efficiently with the following algorithm.

**Algorithm 2.1**
**Input.** $\mathbb{F}_q \equiv \mathbb{F}_p[z]/(\theta(z))$.
**Output.** $\mathcal{E}_{\mathbb{F}_q} = (\mathcal{E}_i(z))_{0 \leqslant i \leqslant \lg \kappa}$.

1. Compute $\mathcal{E}_0(z) := z^p \bmod \theta(z)$ using binary powering.
2. For $i = 1, \ldots, \lg \kappa$, compute $\mathcal{E}_i(z)$ as $\mathcal{E}_{i-1}(z) \circ \mathcal{E}_{i-1}(z) \bmod \theta(z)$.
3. Return $(\mathcal{E}_i(z))_{0 \leqslant i \leqslant \lg \kappa}$.

LEMMA 2.3. *Algorithm 2.1 is correct and runs in time*

$$O(\mathsf{C}_{\mathbb{F}_p}(\kappa) \log \kappa + \mathsf{M}_{\mathbb{F}_p}(\kappa) \log p).$$

**Proof.** We prove the correctness by induction on $i$. For $i = 0$, we clearly have $\mathcal{E}_0(z) = z^p \bmod \theta(z)$. Assume therefore that $i > 0$ and let $h(z)$ be such that

$$\mathcal{E}_{i-1}(z) = z^{p^{2^{i-1}}} + h(z)\,\theta(z).$$

We verify that

$$\begin{aligned}
\mathcal{E}_{i-1}(z) \circ \mathcal{E}_{i-1}(z) &= (z^{p^{2^{i-1}}} + h(z)\,\theta(z)) \circ (z^{p^{2^{i-1}}} + h(z)\,\theta(z)) \bmod \theta(z) \\
&= (z^{p^{2^{i-1}}} + h(z)\,\theta(z)) \circ z^{p^{2^{i-1}}} \bmod \theta(z) \\
&= (z^{p^{2^i}} + (h(z)\,\theta(z))^{p^{2^{i-1}}}) \bmod \theta(z) \\
&= \mathcal{E}_i(z) \bmod \theta(z).
\end{aligned}$$

This completes the correctness proof. The first step takes time $O(\mathsf{M}_{\mathbb{F}_p}(\kappa) \log p)$, whereas the loop requires $\lg \kappa$ modular compositions, whence the complexity bound. □

## 2.2. Iterated absolute pseudo-Frobenius maps

For the efficient application of the absolute pseudo-Frobenius map, we introduce the auxiliary sequence $\mathcal{P}(f) := (\mathcal{P}_i)_{0 \leqslant i \leqslant \lg \kappa}$ with

$$\mathcal{P}_i(x) := x^{p^{2^i}} \bmod f(x), \qquad i = 0, \ldots, \lg \kappa. \tag{2.2}$$

LEMMA 2.4. *Let $\mathcal{P}(f)$ be given. For all $a(x) = a_0 + \cdots + a_{d-1} x^{d-1} \in \mathbb{F}_q[x]_{<d}$ and $0 \leqslant i \leqslant \lg \kappa$, we have*

$$a(x)^{p^{2^i}} \bmod f(x) = \left( \sum_{k=0}^{d-1} a_k^{p^{2^i}} x^k \right) \circ \mathcal{P}_i(x) \bmod f(x). \tag{2.3}$$

*In particular, we may compute $a(x)^{p^{2^i}} \bmod f(x)$ in time $\mathsf{C}_{\mathbb{F}_q}(d) + d\,\Phi_{\mathbb{F}_q} + O(d \log q)$.*

**Proof.** We verify that

$$\begin{aligned} a(x)^{p^{2^i}} \bmod f(x) &= \left(\sum_{k=0}^{d-1} a_k^{p^{2^i}} x^k\right) \circ x^{p^{2^i}} \bmod f(x) \\ &= \left(\sum_{k=0}^{d-1} a_k^{p^{2^i}} x^k\right) \circ \mathcal{P}_i(x) \bmod f(x). \end{aligned}$$

The cost is straightforward from the definitions. □

Once $\mathcal{P}(f)$ has been computed, the pseudo-Frobenius map can be iterated an arbitrary number of times, using the following variant of Lemma 2.2.

LEMMA 2.5. *Let $\mathcal{P}(f)$ be given. For all $a \in \mathbb{A}$ and $e \in \{0, \ldots, \kappa\}$ with binary expansion $e = 2^{i_1} + \cdots + 2^{i_l}$ with $i_1 > \cdots > i_l$, we have*

$$a^{p^e} = (((a^{p^{2^{i_1}}})^{p^{2^{i_2}}})\cdots)^{p^{2^{i_l}}},$$

*that can be computed in time*

$$O((\mathsf{C}_{\mathbb{F}_q}(d) + d\,\Phi_{\mathbb{F}_q}) \log e).$$

**Proof.** The proof is straightforward from Lemma 2.4. □

The auxiliary sequence $\mathcal{P}(f)$ can be computed efficiently as follows.

**Algorithm 2.2**
**Input.** A monic polynomial $f \in \mathbb{F}_q[x]$ of degree $d$.
**Output.** $\mathcal{P}(f) = (\mathcal{P}_i(x))_{0 \leqslant i \leqslant \lg \kappa}$.

1. Compute $\mathcal{P}_0(x) := x^p \bmod f(x)$ using binary powering.
2. For $i = 0, \ldots, \lg \kappa - 1$:
   a. Write $\mathcal{P}_i(x) = \sum_{k=0}^{d-1} a_{i,k} x^k$,
   b. Compute $\mathcal{P}_{i+1}(x)$ as $\left(\sum_{k=0}^{d-1} a_{i,k}^{p^{2^i}} x^k\right) \circ \mathcal{P}_i(x) \bmod f(x)$.
3. Return $(\mathcal{P}_i(x))_{0 \leqslant i \leqslant \lg \kappa}$.

LEMMA 2.6. *Algorithm 2.2 is correct and runs in time*

$$O((\mathsf{C}_{\mathbb{F}_q}(d) + d\,\Phi_{\mathbb{F}_q}) \log \kappa + \mathsf{M}_{\mathbb{F}_q}(d) \log p).$$

**Proof.** Let us prove the correctness by induction on $i$. The result clearly holds for $i = 0$. Assume that it holds for a given $i$ and let $h(x)$ be such that $\mathcal{P}_i(x) = x^{p^{2^i}} + h(x) f(x)$. Then

$$\begin{aligned} \mathcal{P}_{i+1}(x) &= x^{p^{2^{i+1}}} \bmod f(x) \\ &= (x^{p^{2^{i+1}}} + h(x)^{p^{2^i}} f(x)^{p^{2^i}}) \bmod f(x) \\ &= \mathcal{P}_i(x)^{p^{2^i}} \bmod f(x) \\ &= \sum_{k=0}^{d-1} a_{i,k}^{p^{2^i}} (x^k)^{p^{2^i}} \bmod f(x) \\ &= \left(\sum_{k=0}^{d-1} a_{i,k}^{p^{2^i}} x^k\right) \circ x^{p^{2^i}} \bmod f(x). \end{aligned}$$

As to the complexity bound, the binary powering in step 1 takes $O(\mathsf{M}_{\mathbb{F}_q}(d)\log p)$ time. In step 2, we compute $d\lg\kappa$ powers of the form $a_{i,k}^{p^{2^i}}$ and we perform $\lg\kappa$ modular compositions of degree $<d$ over $\mathbb{F}_q$. □

## 2.3. Iterated pseudo-Frobenius

For the efficient application of the pseudo-Frobenius map, we introduce another auxiliary sequence $Q(f) := (Q_i)_{0\leqslant i\leqslant \lg d}$ with

$$Q_i(x) \;:=\; x^{q^{2^i}} \bmod f(x), \qquad i=0,\ldots,\lg d. \tag{2.4}$$

LEMMA 2.7. *Let $Q(f)$ be given. For all $a(x)\in\mathbb{F}_q[x]_{<d}$ and $0\leqslant i\leqslant\lg d$, we have*

$$a(x)^{q^{2^i}} \bmod f(x) \;=\; a(x)\circ Q_i(x) \bmod f(x). \tag{2.5}$$

*In particular, we may compute $a(x)^{q^{2^i}} \bmod f(x)$ in time $\mathsf{C}_{\mathbb{F}_q}(d)+O(d\log q)$.*

**Proof.** The proof is straightforward from the definitions. □

Once $Q(f)$ has been computed, the pseudo-Frobenius map can be iterated an arbitrary number of times by adapting Lemma 2.2, but this will not be needed in the sequel. The sequence $Q(f)$ can be computed efficiently as follows.

**Algorithm 2.3**
**Input.** A monic polynomial $f\in\mathbb{F}_q[x]$ of degree $d$, and $\mathcal{D}(f)$.
**Output.** $Q(f)=(Q_i(x))_{0\leqslant i\leqslant\lg d}$.

1. Compute $Q_0(x)=x^q \bmod f(x)$ using Lemma 2.5.
2. For $i=1,\ldots,\lg d$, compute $Q_i(x)=Q_{i-1}(x)\circ Q_{i-1}(x) \bmod f(x)$.
3. Return $(Q_i(x))_{0\leqslant i\leqslant\lg d}$.

LEMMA 2.8. *Algorithm 2.3 is correct and runs in time*

$$O(\mathsf{C}_{\mathbb{F}_q}(d)\log(d\kappa)+d\,\Phi_{\mathbb{F}_q}\log\kappa).$$

**Proof.** The correctness is proved in a similar way as for Algorithm 2.1. Step 1 takes $O((\mathsf{C}_{\mathbb{F}_q}(d)+d\,\Phi_{\mathbb{F}_q})\log\kappa)$ operations, by Lemma 2.5, whereas step 2 takes $O(\mathsf{C}_{\mathbb{F}_q}(d)\log d)$ operations. □

# 3. PSEUDO-TRACES

Let $q=p^\kappa$ be still as above. Recall that the trace of an element $a\in\mathbb{F}_q$ over $\mathbb{F}_p$, written $\mathrm{Tr}_{\mathbb{F}_q/\mathbb{F}_p}(a)$, is defined as

$$\mathrm{Tr}_{\mathbb{F}_q/\mathbb{F}_p}(a)=a^{p^{\kappa-1}}+a^{p^{\kappa-2}}+\cdots+a^p+a.$$

For a monic, not necessarily irreducible polynomial $f\in\mathbb{F}_q[x]$ of degree $d$, it is customary to consider two similar kinds of maps over $\mathbb{F}_q[x]/(f(x))$, which are called *pseudo-traces*: one over $\mathbb{F}_p$ and one over $\mathbb{F}_q$. In this section, we reformulate fast algorithms for pseudo-traces by Kaltofen and Shoup [23], and make them rely on the data structures from the previous section for the computation of Frobenius maps.

## 3.1. Pseudo-traces over the ground field

Let $a \in \mathbb{F}_q[x]_{<d}$. We define the *pseudo-trace* of $a(x)$ of order $e \geqslant 1$ modulo $f(x)$ over $\mathbb{F}_q$ by

$$\mathrm{Tr}_{q^e;q}(a(x)) \bmod f(x) := a(x)^{q^{e-1}} + \cdots + a(x)^q + a(x) \bmod f(x).$$

We may compute pseudo-traces using the following algorithm:

**Algorithm 3.1**
**Input.** $\mathcal{Q}(f)$, $a(x) \in \mathbb{F}_q[x]_{<d}$, and $e \leqslant d$.
**Output.** $\mathrm{Tr}_{q^e;q}(a(x)) \bmod f(x)$.

1. Let $e = e_l 2^l + e_{l-1} 2^{l-1} + \cdots + e_0$ be the binary expansion of $e$.
2. Let $b_0(x) := a(x)$ and
$$b_i(x) := b_{i-1}(x) + (b_{i-1}(x))^{q^{2^{i-1}}} \bmod f(x), \qquad i = 1, \ldots, l.$$
3. Let $t_{-1}(x) := 0$ and
$$t_i(x) := e_i b_i(x) + (t_{i-1}(x))^{q^{e_i 2^i}} \bmod f(x), \qquad i = 0, \ldots, l.$$
4. Return $t_l(x)$.

LEMMA 3.1. *Algorithm 3.1 is correct and runs in time*

$$O(\mathsf{C}_{\mathbb{F}_q}(d) \log e).$$

**Proof.** By induction on $i$, we verify that

$$\begin{aligned} b_i(x) &= \mathrm{Tr}_{q^{2^i};q}(a(x)) \bmod f(x) \\ t_i(x) &= \mathrm{Tr}_{q^{e_i 2^i + \cdots + e_0};q}(a(x)) \bmod f(x). \end{aligned}$$

The cost follows from Lemma 2.7. □

## 3.2. Absolute pseudo-traces

We define the *absolute pseudo-trace* of $a(x)$ modulo $f(x)$ of order $e$ by

$$\mathrm{Tr}_{p^e;p}(a(x)) \bmod f(x) := a(x)^{p^{e-1}} + \cdots + a(x)^p + a(x) \bmod f(x).$$

We may compute absolute pseudo-traces using the following variant of Algorithm 3.1:

**Algorithm 3.2**
**Input.** $\mathcal{P}(f)$, $a(x) \in \mathbb{F}_q[x]_{<d}$, and $e \leqslant \kappa$.
**Output.** $\mathrm{Tr}_{p^e;p}(a(x)) \bmod f(x)$.

1. Let $e = e_l 2^l + e_{l-1} 2^{l-1} + \cdots + e_0$ be the binary expansion of $e$.
2. Let $b_0(x) := a(x)$ and
$$b_i(x) := b_{i-1}(x) + (b_{i-1}(x))^{p^{2^{i-1}}} \bmod f(x), \qquad i = 1, \ldots, l.$$
3. Let $t_{-1}(x) := 0$ and
$$t_i(x) := e_i b_i(x) + (t_{i-1}(x))^{p^{e_i 2^i}} \bmod f(x), \qquad i = 0, \ldots, l.$$
4. Return $t_l(x)$.

LEMMA 3.2. *Algorithm 3.2 is correct and runs in time*

$$O((\mathsf{C}_{\mathbb{F}_q}(d) + d\,\Phi_{\mathbb{F}_q})\log e).$$

**Proof.** By induction on $i$, we have

$$\begin{aligned} b_i(x) &= \mathrm{Tr}_{p^{2^i};p}(a(x)) \bmod f(x) \\ t_i(x) &= \mathrm{Tr}_{p^{e_i 2^i + \cdots + e_0};p}(a(x)) \bmod f(x). \end{aligned}$$

The cost then follows from Lemma 2.4. □

## 4. POLYNOMIAL FACTORIZATION

We follow the Cantor–Zassenhaus strategy, which subdivides irreducible factorization in $\mathbb{F}_q[x]$ into three consecutive steps:

- the *square-free factorization* decomposes into square-free factors along with their respective multiplicities,
- the *distinct-degree factorization* separates irreducible factors according to their degree,
- the *equal-degree factorization* completely factorizes a polynomial whose irreducible factors have the same degree.

For the distinct-degree factorization, we revisit the "baby-step giant-step" algorithm due to von zur Gathen and Shoup [11, section 6], later improved by Kaltofen and Shoup [24, Algorithm D]. For the equal-degree factorization, we adapt another algorithm due to von zur Gathen and Shoup [11, section 5], while taking advantage of fast modular composition as in [23, Theorem 1]. Throughout this section, we assume that $f \in \mathbb{F}_q[x]$ is the polynomial to be factored and that $f$ is monic of degree $d$.

### 4.1. Square-free factorization

The square-free factorization combines the separable factorization and $p$-th root extractions.

PROPOSITION 4.1. *The square-free factorization of a monic polynomial $f(x) \in \mathbb{F}_q[x]$ of degree $d$ takes time*

$$O(\mathsf{M}_{\mathbb{F}_q}(d)\log d + d\,\mathsf{D}_{\mathbb{F}_q} + d\,\Phi_{\mathbb{F}_q}\log\kappa).$$

**Proof.** Let $a \in \mathbb{F}_q$ and let $k \leqslant \kappa$. The $p^k$-th root of $a$ in $\mathbb{F}_q$ can be computed as $a^{q/p^k} = a^{p^{\kappa-k}}$ in time $O(\Phi_{\mathbb{F}_q}\log\kappa)$ by Lemma 2.2.

The separable factorization of $f$ takes time $O(\mathsf{M}_{\mathbb{F}_q}(d)\log d + d\,\mathsf{D}_{\mathbb{F}_q})$; see [27]. This yields

$$f(x) = \prod_{i=1}^{r} f_i\big(x^{p^{k_i}}\big)^{m_i},$$

where the $f_i$ are monic and separable, the $f_i\big(x^{p^{k_i}}\big)$ are pairwise coprime, and $p$ does not divide the $m_i$.

In order to deduce the square-free factorization of $f$ it remains to extract the $p^{k_i}$-th roots of the coefficients of $f_i$, for $i = 1, \ldots, r$. The cost of these extractions is bounded by

$$O\left(\sum_{i=1}^{r} \deg f_i\,\Phi_{\mathbb{F}_q}\log\kappa\right) = O(d\,\Phi_{\mathbb{F}_q}\log\kappa).$$

□

## 4.2. Distinct-degree factorization

In this subsection $f$ is assumed to be monic and square-free. The distinct-degree factorization is a partial factorization $g_1 \cdots g_d$ of $f$ where $g_i$ is the product of the monic irreducible factors of $f$ of degree $i$. The following algorithm exploits the property that $\gcd\left(x^{q^i}-x, f(x)\right)$ is the product of the irreducible factors of $f$ of a degree that divides $i$. The "baby-step giant-step" paradigm is used in order to avoid the naive computation of the $x^{q^i} \bmod f(x)$ in sequence for $i=1,\ldots,d$. As a useful feature, our algorithm only computes the irreducible factors up to a given degree $D$.

**Algorithm 4.1**

**Input.** $f(x) \in \mathbb{F}_q[x]$ monic and square-free of degree $d \geqslant 1$, $D \leqslant d$, $\mathcal{P}(f)$.

**Output.** $g_1,\ldots,g_D$ such that $g_i$ is the product of the irreducible factors of $f$ of degree $i$.

1. Let $\delta := \lfloor D^{0.5} \rfloor$.
2. Compute $x^q \bmod f(x)$ by using $\mathcal{P}(f)$.
3. Compute $x^{q^k} \bmod f(x)$ for $k=2,\ldots,\delta$, via modular compositions.
4. Compute $x^{q^{l\delta}} \bmod f(x)$ for $l=2,\ldots,\lceil D/\delta \rceil$, via modular compositions.
5. Compute $A(y) := \prod_{k=0}^{\delta-1}\left(y - x^{q^k}\right) \bmod f(x)$, where $y$ denotes a new variable.
6. Compute $a_l(x) := A\left(x^{q^{l\delta}}\right) \bmod f(x)$ for $l=1,\ldots,\lceil D/\delta \rceil$.
7. Set $b := f$. For $l=1,\ldots,\lceil D/\delta \rceil$ do:
   Compute $h_l := \gcd(a_l, b)$, $b := b/h_l$.
8. For $k=0,\ldots,\delta-1$, compute $x^{q^k} \bmod h_i$ for $i=1,\ldots,\lceil D/\delta \rceil$.
9. For $l=1,\ldots,\lceil D/\delta \rceil$, compute $x^{q^{l\delta}} \bmod h_i$ for $i=1,\ldots,\lceil D/\delta \rceil$.
10. For $l=1,\ldots,\lceil D/\delta \rceil$ do:
    Set $b := h_l$.
    For $k$ from $\delta-1$ down to 0 do:
    Compute $g_{l\delta-k} := \gcd\left(b, x^{q^{l\delta}} - x^{q^k} \bmod h_l\right)$, $b := b/g_{l\delta-k}$.
11. Return $g_1,\ldots,g_D$.

PROPOSITION 4.2. *Algorithm 4.1 is correct and takes time*

$$O(\mathsf{C}_{\mathbb{F}_q}(d;\lfloor D^{0.5}\rfloor+1) + D^{0.5}(\mathsf{M}_{\mathbb{F}_q}(d)\log d + d\,\mathsf{D}_{\mathbb{F}_q}) + (\mathsf{C}_{\mathbb{F}_q}(d) + d\,\Phi_{\mathbb{F}_q})\log\kappa).$$

**Proof.** First note that any positive integer $i \leqslant D$ writes uniquely as $i = l\delta - k$ with $0 \leqslant k < \delta$ and $1 \leqslant l \leqslant \lceil D/\delta \rceil \leqslant \lceil D^{0.5} \rceil + 1$. Then note that

$$\begin{aligned} a_l &= \prod_{k=0}^{\delta-1}\left(x^{q^{l\delta}} - x^{q^k}\right) \bmod f(x) \\ &= \prod_{k=0}^{\delta-1}\left(x^{q^{l\delta-k}} - x\right)^{q^k} \bmod f(x), \end{aligned}$$

so $h_l = g_{l\delta-(\delta-1)} \cdots g_{l\delta}$ for $l=1,\ldots,\lceil D/\delta \rceil$. This shows that $g_1,\ldots,g_l$ are computed correctly.

Lemma 2.5 allows us to perform step 2 in time

$$O((\mathsf{C}_{\mathbb{F}_q}(d) + d\,\Phi_{\mathbb{F}_q})\log\kappa).$$

Step 3 requires $\delta-1$ modular compositions of the form $a(x)\circ\beta(x) \bmod f(x)$ for which $\beta(x) := x^q \bmod f(x)$ is fixed. The same holds for step 4, this time with $\beta(x) := x^{q^\delta} \bmod f(x)$. Consequently, steps 3 and 4 can be done in time

$$O(\mathsf{C}_{\mathbb{F}_q}(d;\lfloor D^{0.5}\rfloor+1)).$$

For steps 5 and 6, we use the classical "divide and conquer" technique based on "subproduct trees", and Kronecker substitution for products in $\mathbb{F}_q[x,y]$; see [9, chapter 10]. These steps then require $O(\mathsf{M}_{\mathbb{F}_q}(D^{0.5}d)\log D)$ operations. Our assumption on $\mathsf{M}_{\mathbb{F}_q}$ yields $O(\mathsf{M}_{\mathbb{F}_q}(D^{0.5}d)\log D) = O(D^{0.5}\mathsf{M}_{\mathbb{F}_q}(d)\log d)$. Step 7 incurs

$$O(D^{0.5}\mathsf{M}_{\mathbb{F}_q}(d)\log d + D^{0.5}d\,\mathsf{D}_{\mathbb{F}_q})$$

operations by means of the half-gcd algorithm.

By construction, the $h_l$ are pairwise coprime and their product equals $f$. Steps 8 and 9 take $O(D^{0.5}\mathsf{M}_{\mathbb{F}_q}(d)\log d)$ operations, by applying the fast multi-remainder algorithm [9, chapter 10] to the results of steps 3 and 4. Finally, the cost of step 10 is bounded by

$$O\left(\sum_{l=1}^{\lceil D/\delta\rceil} D^{0.5}\,(\mathsf{M}_{\mathbb{F}_q}(\deg h_l)\log(\deg h_l) + \deg h_l\,\mathsf{D}_{\mathbb{F}_q})\right) = O(D^{0.5}\,(\mathsf{M}_{\mathbb{F}_q}(d)\log d + d\,\mathsf{D}_{\mathbb{F}_q})). \quad \square$$

## 4.3. Equal-degree factorization

We now turn to the factorization of a polynomial $f\in\mathbb{F}_q[x]$ having all its factors of the same known degree $\delta$. This stage involves randomization of Las Vegas type: the algorithm always returns a correct answer, but the running time is a random variable.

**Algorithm 4.2**

**Input.** A monic, square-free polynomial $f(x)\in\mathbb{F}_q[x]$ of degree $d$, which is the product of $d/\delta$ irreducible factors of degree $\delta$, as well as $\mathcal{P}(f)$ and $\mathcal{Q}(f)$.

**Output.** The irreducible factors of $f$.

1. If $d=\delta$ then return $f$. Otherwise set $\mathcal{I} := \{-1,0,1\}$ if $p\neq 2$, or $\mathcal{I} := \{0,1\}$ if $p=2$.
2. Take $g$ at random in $\mathbb{F}_q[x]_{<d}$.
3. Compute $h_q := \mathrm{Tr}_{q^\delta;q}(g) \bmod f$ by Algorithm 3.1.
4. Compute $h_p := \mathrm{Tr}_{p^\kappa;p}(h_q) \bmod f$ by Algorithm 3.2.
5. If $p\neq 2$, then compute $h := h_p^{\frac{p-1}{2}} \bmod f$. Otherwise, set $h := h_p$.
6. Compute $f_0 := \gcd(f,h)$, $f_1 := \gcd(f,h-1)$, and $f_{-1} := f/(f_0 f_1)$ if $p\neq 2$.
7. Compute $\mathcal{P}(f_i)$ as $\mathcal{P}(f) \bmod f_i$, $\mathcal{Q}(f_i)$ as the $\lg(\deg f_i)+1$ first entries of $\mathcal{Q}(f) \bmod f_i$, for $i\in\mathcal{I}$.
8. For $i\in\mathcal{I}$ call recursively the algorithm with input $f_i$, $\mathcal{P}(f_i)$ and $\mathcal{Q}(f_i)$.
9. Return the union of the irreducible factors of $f_i$ for $i\in\mathcal{I}$.

PROPOSITION 4.3. *Algorithm 4.1 is correct and takes expected time*

$$O((\mathsf{C}_{\mathbb{F}_q}(d)\log(\delta\kappa) + \mathsf{M}_{\mathbb{F}_q}(d)\log(d\kappa p) + d\,\mathsf{D}_{\mathbb{F}_q} + d\,\Phi_{\mathbb{F}_q}\log\kappa)\log(d/\delta)). \tag{4.1}$$

**Proof.** The proof is well known. For completeness, we repeat the main arguments. Let $\varphi_1, \ldots, \varphi_r$ with $r := d/\delta$ be the irreducible factors of $f$. The Chinese remainder theorem yields an isomorphism

$$\chi\colon\ \mathbb{F}_q[x]/(f) \longrightarrow \mathbb{F}_q[x]/(\varphi_1) \times \cdots \times \mathbb{F}_q[x]/(\varphi_r),$$

where each $\mathbb{F}_q[x]/(\varphi_i)$ is isomorphic to $\mathbb{F}_{q^\delta}$. For any $g$ in $\mathbb{F}_q[x]$, let

$$(g_1 \bmod \varphi_1, \ldots, g_r \bmod \varphi_r) := \chi(g \bmod f).$$

Now

$$\chi(h_q \bmod f) = \chi(\mathrm{Tr}_{q^\delta;q}(g) \bmod f) = \left(\mathrm{Tr}_{\mathbb{F}_{q^\delta}/\mathbb{F}_q}(g_1) \bmod \varphi_1, \ldots, \mathrm{Tr}_{\mathbb{F}_{q^\delta}/\mathbb{F}_q}(g_r) \bmod \varphi_r\right)$$

and

$$\begin{aligned}\chi(h_p \bmod f) &= \left(\mathrm{Tr}_{\mathbb{F}_q/\mathbb{F}_p}\left(\mathrm{Tr}_{\mathbb{F}_{q^\delta}/\mathbb{F}_q}(g_1) \bmod \varphi_1\right), \ldots, \mathrm{Tr}_{\mathbb{F}_q/\mathbb{F}_p}\left(\mathrm{Tr}_{\mathbb{F}_{q^\delta}/\mathbb{F}_q}(g_r) \bmod \varphi_r\right)\right)\\ &= \left(\mathrm{Tr}_{\mathbb{F}_{q^\delta}/\mathbb{F}_p}(g_1 \bmod \varphi_1), \ldots, \mathrm{Tr}_{\mathbb{F}_{q^\delta}/\mathbb{F}_p}(g_r \bmod \varphi_r)\right),\end{aligned}$$

where each $\mathrm{Tr}_{\mathbb{F}_{q^\delta}/\mathbb{F}_p}(g_i \bmod \varphi_i)$ belongs to $\mathbb{F}_p$ regarded as the prime subfield of $\mathbb{F}_q[x]/(\varphi_i)$. Hence $\chi(h)$ is a vector $(b_1, \ldots, b_r)$ in $\mathcal{I}^r$, and $f_i$ is the product of the $\varphi_j$ with $b_j = \epsilon$, for $\epsilon \in \mathcal{I}$.

Let $i \in \{1, \ldots, r\}$ be a fixed index. If $p \neq 2$, then the probability that $b_i = 0$ is $1/p$, the probability that $b_i = -1$ is $(p-1)/(2p)$, and the probability that $b_i = 1$ is $(p-1)/(2p)$. If $p = 2$, then the probability that $b_i = 0$ is $1/2$, the probability that $b_i = 1$ is $1/2$.

We now apply [11, Lemma 4.1(i)] with $\ell = |\mathcal{I}|$ and $w = 1/2$. This lemma concerns the probability analysis of a game of balls and bins where the bins are $f_i$ for $i \in \mathcal{I}$ and the balls are the irreducible factors $\varphi_1, \ldots, \varphi_r$ of $f$. The lemma implies that the expected depth of the recursive calls is $O(\log r) = O(\log(d/\delta))$. Other proofs may be found in [9, chapter 14, Exercise 14.16], [35, chapter 20, section 4], or [11, sections 3 and 4].

Step 3 takes $O(\mathsf{C}_{\mathbb{F}_q}(d) \log \delta)$ operations, by Lemma 3.1. Step 4 takes $O((\mathsf{C}_{\mathbb{F}_q}(d) + d\,\Phi_{\mathbb{F}_q}) \log \kappa)$ operations, by Lemma 3.2. Step 5 requires $O(\mathsf{M}_{\mathbb{F}_q}(d) \log p)$ further operations. The rest takes $O(\mathsf{M}_{\mathbb{F}_q}(d) \log(d\kappa) + d\,\mathsf{D}_{\mathbb{F}_q})$ operations. □

Cantor and Zassenhaus' original algorithm [5] uses the map $h \mapsto h^{\frac{q^\delta - 1}{2}} \bmod f$ instead of pseudo-traces whenever $p \neq 2$. For $p = 2$ it uses a slightly different map combined with an occasional quadratic extension of $\mathbb{F}_q$. The use of pseudo-traces appeared in early works by McEliece [9, notes of chapter 14]. The modern presentation is due to [11]. Our presentation has the advantage to distinguish the pseudo-traces over $\mathbb{F}_q$ from those over $\mathbb{F}_p$, and to avoid recomputing $\mathcal{P}(f)$ and $\mathcal{Q}(f)$ during recursive calls.

**Remark 4.1.** If $\mathsf{C}_{\mathbb{F}_q}(d) = \Omega(d^{1+\eta} \log q)$ for some $\eta > 0$, then $\mathsf{C}_{\mathbb{F}_q}(d) \log(\delta\kappa)$ does not need to be multiplied by $\log(d/\delta)$ in (4.1); see [11, Lemma 4.1(ii)].

## 4.4. Irreducible factorization

We are now ready to summarize the main complexity bounds for an abstract field $\mathbb{F}_q$, in terms of the cost functions from section 1.1.

THEOREM 4.1. *The computation of the irreducible factors of degree $\leqslant D$ of a polynomial of degree $d$ in $\mathbb{F}_q[x]$ can be done in expected time*

$$\begin{aligned}O\Big((D^{0.5} + \log(d\kappa) \log d)\, \mathsf{C}_{\mathbb{F}_q}(d) + (D^{0.5} + \log(d\kappa p))\, \mathsf{M}_{\mathbb{F}_q}(d) \log d\\ + (D^{0.5} + \log d)\, d\, \mathsf{D}_{\mathbb{F}_q} + d\,\Phi_{\mathbb{F}_q} \log d \log \kappa\Big).\end{aligned}$$

**Proof.** This bound follows by combining Lemmas 2.6 and 2.8, Propositions 4.1, 4.2, and 4.3. □

Following [9, Corollary 14.35] from [33, section 6], a polynomial $f \in \mathbb{F}_q[x]$ of degree $d \geqslant 1$ is irreducible if, and only if, $f$ divides $x^{q^d} - x$ and $\gcd\left(x^{q^{d/t}} - x, f\right) = 1$ for all prime divisors $t$ of $d$. This technique was previously used in [32] over prime fields.

THEOREM 4.2. *A polynomial of degree $d$ in $\mathbb{F}_q[x]$ can be tested to be irreducible in time*

$$O(\mathsf{C}_{\mathbb{F}_q}(d)\,(\log d \log\log d + \log\kappa) + \mathsf{M}_{\mathbb{F}_q}(d)\,(\log^2 d + \log p) + d\,\mathsf{D}_{\mathbb{F}_q}\log d + d\,\Phi_{\mathbb{F}_q}\log\kappa).$$

**Proof.** Computing the prime factorization $d = p_1^{m_1} \cdots p_r^{m_r}$ takes negligible time $o(d)$. On the other hand, we can compute $\mathcal{D}(f)$ in time

$$O((\mathsf{C}_{\mathbb{F}_q}(d) + d\,\Phi_{\mathbb{F}_q})\log\kappa + \mathsf{M}_{\mathbb{F}_q}(d)\log p),$$

by Lemma 2.6. Then $x^q \bmod f(x)$ can be obtained in time

$$O((\mathsf{C}_{\mathbb{F}_q}(d) + d\,\Phi_{\mathbb{F}_q})\log\kappa),$$

by Lemma 2.5.

The "divide and conquer" strategy of [33, Lemma 6.1] allows us to compute $x^{q^{d/p_1}}, \ldots, x^{q^{d/p_r}}$ in time $O(\mathsf{C}_{\mathbb{F}_q}(d)\log d \log r)$; see the proof of [33, Theorem 6.2]. Finally each gcd takes $O(\mathsf{M}_{\mathbb{F}_q}(d)\log d + d\,\mathsf{D}_{\mathbb{F}_q})$ operations. □

## 4.5. Rabin's strategy to save pseudo-trace computations

We finish this section with a digression on known optimizations for the equal-degree factorization algorithm that will not be used in the rest of the paper. These optimizations are based on a randomization strategy due to Rabin [32] that saves pseudo-norm and pseudo-trace computations. Here we focus on the case $p \neq 2$; in the case when $p = 2$, similar but slightly different formulas can be used [1, 32]. A concise presentation of Rabin's method is given in [9, Exercise 14.17], but for pseudo-norms instead of pseudo-traces. For this reason, we briefly repeat the main arguments. We follow the notation of Algorithm 4.2.

Assume that $f \in \mathbb{F}_q[x]$ is monic and square-free of degree $d$ and a product of $r$ monic irreducible factors $\varphi_1, \ldots, \varphi_r$ of degree $\delta = d/r$. Consider a polynomial $h \in \mathbb{F}_q[x]$ such that $h \bmod \varphi_i \in \mathbb{F}_p$ for $i = 1, \ldots, r$. We say that $h$ *separates* the irreducible factors of $f$ if $h \bmod \varphi_i \neq h \bmod \varphi_j$ for all $i \neq j$.

**Algorithm 4.3**

**Input.** A monic, square-free polynomial $f(x) \in \mathbb{F}_q[x]$ of degree $d$, which is the product of $r$ monic irreducible factors of degree $\delta = d/r$, a polynomial $h_p \in \mathbb{F}_q[x]$ with $h \bmod \varphi_i \in \mathbb{F}_p$ for $i = 1, \ldots, r$, and an integer $t \geqslant 0$.

**Output.** A partial factorization of $f$.

1. If $d = \delta$ or $t = 0$, then return $f$.
2. Take $\tau$ at random in $\mathbb{F}_p$.
3. Compute $h := (h_p + \tau)^{\frac{p-1}{2}} \bmod f$.

4. Compute $f_0 := \gcd(f, h)$, $f_1 := \gcd(f, h-1)$, and $f_{-1} := f/(f_0 f_1)$.
5. For $i = -1, 0, 1$, call recursively the algorithm with input $f_i$, $h_p \bmod f_i$, and $t-1$.
6. Return the union of the factors of $f_{-1}, f_0, f_1$ collected during step 5.

PROPOSITION 4.4. *Algorithm 4.3 is correct and takes time*

$$O((\mathsf{M}_{\mathbb{F}_q}(d)\log(dp) + d\,\mathsf{D}_{\mathbb{F}_q})\,t).$$

*In addition the following assertions hold:*

*i. For $g$ taken at random in $\mathbb{F}_q[x]_{<d}$, $h_p := \mathrm{Tr}_{p^\kappa;p}(\mathrm{Tr}_{q^\delta;q}(g)) \bmod f$ does not separate the factors of $f$ with probability $\leqslant \binom{r}{2}\frac{1}{p}$.*

*ii. If $h_p$ separates the irreducible factors of $f$, then Algorithm 4.3, called with $t$ such that $2^t \geqslant \frac{1}{\eta}\binom{r}{2}$, returns all the irreducible factors of $f$ with probability $\geqslant 1-\eta$.*

**Proof.** The proof of the complexity bound is straightforward. A random $g$ yields $h_p$ such that $h_p \bmod \varphi_i = h_p \bmod \varphi_j$ for $i \neq j$ with probability $1/p$. Therefore a random $\tau$ yields a polynomial $h_p$ that does not separate the irreducible factors of $f$ with probability at most $\binom{r}{2}/p$. That proves assertion (*i*).

Now assume that $h_p$ separates the irreducible factors of $f$. Given $i \neq j$ in $\{1, \ldots, r\}$, we have

$$(h_p + \tau)^{\frac{p-1}{2}} \bmod \varphi_i = (h_p + \tau)^{\frac{p-1}{2}} \bmod \varphi_j \tag{4.2}$$

for at most $\frac{p-1}{2} - 1$ values of $\tau$. With $\tau$ taken at random in $\mathbb{F}_p$, the probability that (4.2) does not hold is therefore at least $1/2$.

Let $P(t)$ denote the probability that all the irreducible factors are not found after the call of Algorithm 4.3 with input $t$. There exist $i \neq j$ such that (4.2) holds for $t$ random values of $\tau$ with probability at most $2^{-t}$. Considering the $\binom{r}{2}$ possible pairs $\{i, j\}$, we obtain $P(t) \leqslant \binom{r}{2}/2^t$. □

We may benefit from Rabin's strategy within Algorithm 4.2 as follows, whenever $p \geqslant \frac{1}{\epsilon}\binom{r}{2}$. A polynomial $h_p$ as in Proposition 4.4(*i*) separates the factors of $f$ with probability $\geqslant 1-\epsilon$. We call Algorithm 4.3 with the first value of $t$ such that $2^t \geqslant \frac{1}{\eta}\binom{r}{2}$, so the irreducible factors of $f$ are found with probability $\geqslant 1-\eta$. When $\epsilon$ and $\eta$ can be taken sufficiently small, we derive a similar complexity bound as in Proposition 4.3, but where the factor $\log(d/\delta)$ does not apply to the terms $d\,\Phi_{\mathbb{F}_q}\log\kappa$ and $\mathsf{C}_{\mathbb{F}_q}(d)\log(\delta\kappa)$, which correspond to the costs of steps 3 and 4 of Algorithm 4.2.

If $p$ is too small to find a suitable value for $\epsilon$, then we may appeal to Rabin's strategy a few rounds in order to benefit from more splittings with a single pseudo-trace over $\mathbb{F}_p$. If $p$ is actually too small for this approach, then we may consider the case $q \geqslant \frac{1}{\epsilon}\binom{r}{2}$ and apply Rabin's strategy over $\mathbb{F}_q$ instead of $\mathbb{F}_p$. More precisely, from $h_q$ and a random $\tau \in \mathbb{F}_q$ we compute

$$h_p := \mathrm{Tr}_{p^\kappa;p}(h_q + \tau) \bmod f,$$

then $h := h_p^{\frac{p-1}{2}} \bmod f$, and obtain the splitting $\gcd(h, f)$, $\gcd(h, f-1)$, $\gcd(h, f+1)$ on which to recurse. This approach yields a complexity bound similar to the one from Proposition 4.3, but where the factor $\log(d/\delta)$ does not apply to the term $\mathsf{C}_{\mathbb{F}_q}(d)\log(\delta\kappa)$. This latter term corresponds to the cost of step 3 of Algorithm 4.2.

# 5. THEORETICAL COMPLEXITY BOUNDS

This section first draws corollaries from section 4.4, which rely on Kedlaya and Umans' algorithm for modular composition. Note however that it seems unlikely that this algorithm can be implemented in a way that makes it efficient for practical purposes: see [19, Conclusion]. We first consider the case when $\mathbb{F}_q$ is a primitive extension over $\mathbb{F}_p$ and then the more general case when $\mathbb{F}_q$ is given via a "triangular tower".

## 5.1. Factoring over primitive extensions

Assume that $\mathbb{F}_q \equiv \mathbb{F}_p[z]/(\theta(z))$ is a primitive extension of $\mathbb{F}_p$. Then we may take:

- $\mathsf{M}_{\mathbb{F}_q}(d) = O(d \log q \log(d \log q)\, 4^{\log^*(d \log q)}) = \tilde{O}(d \log q)$: see [13]. Under a plausible number theoretic conjecture, one even has $\mathsf{M}_{\mathbb{F}_q}(d) = O(d \log q \log(d \log q))$ [14].
- $\mathsf{D}_{\mathbb{F}_q} = O(\mathsf{M}_{\mathbb{F}_p}(\kappa) \log \kappa) = \tilde{O}(\log q)$: see [9].
- From [25, 26],
$$\mathsf{C}_{\mathbb{F}_q}(d) = d^{1+\epsilon(d)}\, \tilde{O}(\log q), \tag{5.1}$$
where $\epsilon = o(1)$; The refined bound with
$$\epsilon(d) = O\left(\left(\frac{\log\log d}{\log d}\right)^{1/2}\right)$$
is proved in [19, section 6].

COROLLARY 5.1. *Let $\mathbb{F}_q \equiv \mathbb{F}_p[z]/(\theta(z))$ be as above. The computation of the irreducible factors of degree $\leqslant D$ of a polynomial of degree $d$ in $\mathbb{F}_q[x]$ takes an expected time*
$$(D^{0.5} d^{\epsilon(d)} + \kappa^{\epsilon(\kappa)} + \log p)\, \tilde{O}(d \log q).$$

**Proof.** By means of (5.1) the precomputation involved in Lemma 2.3 takes
$$O(\mathsf{C}_{\mathbb{F}_p}(\kappa) \log \kappa + \mathsf{M}_{\mathbb{F}_p}(\kappa) \log p) = (\kappa^{\epsilon(\kappa)} + \log p)\, \tilde{O}(\log q).$$
Then $\Phi_{\mathbb{F}_q} = \kappa^{\epsilon(\kappa)}\, \tilde{O}(\log q)$ holds by Lemma 2.1. So the bound follows from Theorem 4.1. □

COROLLARY 5.2. *Let $\mathbb{F}_q \equiv \mathbb{F}_p[z]/(\theta(z))$ be as above. A polynomial of degree $d$ in $\mathbb{F}_q[x]$ can be tested to be irreducible in time*
$$(d^{\epsilon(d)} + \kappa^{\epsilon(\kappa)} + \log p)\, \tilde{O}(d \log q).$$

**Proof.** This follows from Theorem 4.2, in a similar way as above. □

## 5.2. Factoring over towers of finite fields

Now we examine the case where $\mathbb{F}_q \equiv \mathbb{K}_t$, where $(\mathbb{K}_i)_{i \leqslant t}$ a *triangular tower* of height $t$ of field extensions $\mathbb{K}_i \supseteq \mathbb{F}_p$ such that $\mathbb{K}_0 := \mathbb{F}_p$ and
$$\mathbb{K}_i \equiv \mathbb{K}_{i-1}[z_i]/(\mu_i(z_i)), \qquad i = 1, \ldots, t,$$
for irreducible polynomials $\mu_i \in \mathbb{K}_{i-1}[z_i]$. We write $m_i := \deg \mu_i$, so $\kappa := m_1 \cdots m_t$, and assume that $m_i \geqslant 2$ for $i = 1, \ldots, t$.

Using [31, Theorem 1.2], we will describe how to compute an isomorphic primitive representation of $\mathbb{F}_q$, and how to compute the corresponding conversions. This will allow us to apply the results from section 5.1. In this subsection, we assume the boolean RAM model instead of the Turing model, in order to use the results from [31].

COROLLARY 5.3. *Fix $\varepsilon > 0$. Let $\mathbb{F}_q \equiv \mathbb{K}_t$ be given as above for a triangular tower $(\mathbb{K}_i)_{i \leqslant t}$ and assume that $p > \kappa$. Then the computation of the irreducible factors of degree $\leqslant D$ of a polynomial $f$ of degree $d$ in $\mathbb{F}_q[x]$ takes an expected time*

$$(D^{0.5} d^{\varepsilon} + \kappa^{\varepsilon} + \log p)\, \tilde{O}(d \log q).$$

**Proof.** The number of monic irreducible polynomials of degree $\kappa$ over $\mathbb{F}_p$ is $\geqslant \frac{p^\kappa - 2p^{\kappa/2}}{\kappa}$; see for instance [9, Lemma 14.38]. Therefore the number of elements $\gamma$ in $\mathbb{F}_q$ that generate $\mathbb{F}_q$ is $\geqslant p^\kappa - 2p^{\kappa/2}$. The probability to pick up a generator of $\mathbb{F}_q$ over $\mathbb{F}_p$ is uniformly lower bounded, since $p > \kappa \geqslant 2$.

The assumption $p > \kappa$ allows us to apply [31, Theorem 1.2] to the following data:

- The tower $(\mathbb{K}_i)_{i \leqslant t}$ extended with
$$\mathbb{K}_{t+1} \equiv \mathbb{K}_t[z_{t+1}] / (z_{t+1} - \gamma),$$
where $\gamma$ is an element picked up randomly in $\mathbb{K}_t$;
- The "target order" $z_1 > z_t > \cdots > z_{t+1}$.

When $\gamma$ generates $\mathbb{F}_q$ over $\mathbb{F}_p$ we obtain an isomorphic primitive element presentation of $\mathbb{K}_t$ of the form $\mathbb{F}_p[\gamma] \equiv \mathbb{F}_p[t] / (\theta(t))$, where $\theta$ denotes the defining polynomial of $\gamma$ over $\mathbb{F}_p$. If $\gamma$ is not primitive then the algorithm underlying [31, Theorem 1.2] is able to detect it. In all cases, the time to construct $\mathbb{F}_p[\gamma]$ is $\kappa^{\varepsilon} \tilde{O}(\log q)$. If $\gamma$ is primitive, then [31, Theorem 1.2] also ensures conversions between $\mathbb{K}_{t+1}$ and $\mathbb{F}_p[\gamma]$ in time $\kappa^{\varepsilon} \tilde{O}(\log q)$. Consequently the result follows from Corollary 5.1. □

## 6. PRACTICAL COMPLEXITY BOUNDS

An alternative approach for modular composition over $\mathbb{F}_q$ was proposed in [16]. The approach is only efficient when the extension degree $\kappa$ over $\mathbb{F}_p$ is composite. If $\kappa$ is smooth and if mild technical conditions are satisfied, then it is even quasi-optimal. It also does not rely on the Kedlaya–Umans algorithm and we expect it to be useful in practice for large composite $\kappa$.

The main approach from [16] can be applied in several ways. As in [16], we will first consider the most general case when $\mathbb{F}_q$ is presented via a triangular tower. In that case, it is now possible to benefit from accelerated tower arithmetic that was designed in [17]. We next examine several types of "primitive towers" for which additional speed-ups are possible.

In order to apply the results from [16, 17], all complexity bounds in this section assume the boolean RAM model instead of the Turing model.

### 6.1. Factoring over triangular towers

As in section 5.2, a *triangular tower* over $\mathbb{F}_p$ is a tower of algebraic extensions

$$\mathbb{F}_p \equiv \mathbb{K}_0 \subset \mathbb{K}_1 \subset \cdots \subset \mathbb{K}_t \equiv \mathbb{F}_q.$$

such that $\mathbb{K}_{i+1}$ is presented as a primitive extension over $\mathbb{K}_i$. In other words, for $i = 1, \ldots, t$, we have

$$\mathbb{K}_i \cong \mathbb{K}_{i-1}[z_i] / (\mu_i(z_i))$$

for some monic irreducible polynomial $\mu_i \in \mathbb{K}_i[z_i]$ of degree $m_i \geqslant 2$. Alternatively, each $\mathbb{K}_i$ can be presented directly over $\mathbb{K}_0$ as a quotient $\mathbb{K}_0[x_1, \ldots, x_i] / (\breve{\mu}_1, \ldots, \breve{\mu}_i)$, where $\breve{\mu}_i \in \mathbb{K}_0[x_1, \ldots, x_i]$ is monic of degree $m_i$ in $x_i$ and of degree $< m_j$ in $x_j$ for each $j < i$. Triangular towers have the advantage that $\mathbb{K}_{i-1}$ is naturally embedded in $\mathbb{K}_i$ for each $i$. We set

$$\bar{m} := \max(m_1, \ldots, m_t).$$

### 6.1.1. Basic arithmetic

PROPOSITION 6.1. *There exists a function*

$$\epsilon(\kappa) = O\left(\frac{1}{\sqrt{\log \kappa}}\right),$$

*such that $\kappa^{\epsilon(\kappa)}$ is a non-decreasing function of $\kappa$, and*

$$\begin{aligned} \mathsf{M}_{\mathbb{K}_t}(d) &= \kappa^{\epsilon(\kappa)} \tilde{O}(d \log q) \\ \mathsf{D}_{\mathbb{K}_t} &= \kappa^{\epsilon(\kappa)} \tilde{O}(\log q). \end{aligned}$$

**Proof.** If $p > \binom{\kappa}{2}$, then [17, Proposition 2.7 and Corollary 4.11] imply

$$\begin{aligned} \mathsf{M}_{\mathbb{K}_t}(d) &= \kappa^{\epsilon(\kappa)} \tilde{O}(d \log q) \\ \mathsf{D}_{\mathbb{K}_t} &= \kappa^{\epsilon(\kappa)} \tilde{O}(\log q). \end{aligned}$$

Now consider the case when $p \leqslant \binom{\kappa}{2}$. Since $p^\kappa \geqslant \binom{\kappa}{2}$ there exists a smallest integer $l \leqslant t$ such that $p^{m_1 \cdots m_l} \geqslant \binom{\kappa}{2}$, and

$$l = O(\log \log \kappa).$$

For any constant $C > 1$ we note that $C^l = \log^{O(1)} \kappa$, so [17, Propositions 2.4 and 2.7] yield

$$\begin{aligned} \mathsf{M}_{\mathbb{K}_l}(d) &= \tilde{O}(d m_1 \cdots m_l \log p) \log^{O(1)} \kappa \\ \mathsf{D}_{\mathbb{K}_l} &= \tilde{O}(m_1 \cdots m_l \log p) \log^{O(1)} \kappa. \end{aligned}$$

On the other hand, applying [17, Proposition 2.7 and Corollary 4.11] to the sub-tower

$$\mathbb{K}_l \subset \cdots \subset \mathbb{K}_t,$$

we obtain

$$\begin{aligned} \mathsf{M}_{\mathbb{K}_t}(d) &= (m_{l+1} \cdots m_t)^{\epsilon(m_{l+1} \cdots m_t)} \tilde{O}(d m_{l+1} \cdots m_t (\mathsf{M}_{\mathbb{K}_l}(1) + \mathsf{D}_{\mathbb{K}_l})) \\ &= (m_{l+1} \cdots m_t)^{\epsilon(m_{l+1} \cdots m_t)} \tilde{O}(d \kappa \log p) \\ &= \kappa^{\epsilon(\kappa)} \tilde{O}(d \log q) \\ \mathsf{D}_{\mathbb{K}_t} &= \mathsf{D}_{\mathbb{K}_l} + (m_{l+1} \cdots m_t)^{\epsilon(m_{l+1} \cdots m_t)} \tilde{O}(m_{l+1} \cdots m_t \mathsf{M}_{\mathbb{K}_l}(1)) \\ &= (m_{l+1} \cdots m_t)^{\epsilon(m_{l+1} \cdots m_t)} \tilde{O}(\kappa \log p) \\ &= \kappa^{\epsilon(\kappa)} \tilde{O}(\log q). \end{aligned}$$

□

### 6.1.2. Frobenius maps

In order to compute iterated Frobenius maps we extend the construction from section 2.1. For this purpose we introduce the following auxiliary sequences for $k = 1, \ldots, t$:

$$\mathcal{E}_{\mathbb{K}_k} := \left( z_k^{p^{2^i}} \bmod \mu_k(z_k) \right)_{0 \leqslant i \leqslant \lg \kappa}.$$

Since we wish to avoid relying on the Kedlaya–Umans algorithm, the best available algorithm for modular composition is based on the "baby-step giant-step" method [30]. It yields the following complexity bound:

$$\mathsf{C}_{\mathbb{F}_q}(d) \;=\; O(d^{\omega}\mathsf{M}_{\mathbb{F}_q}(1)),$$

where $\omega > 1.5$ is a constant such that the product of a $d \times \sqrt{d}$ matrix by a $\sqrt{d} \times \sqrt{d}$ matrix takes $O(d^{\omega})$ operations; the best known theoretical bound is $\omega < 1.667$ [20, Theorem 10.1]. In practice, one usually assumes $\omega = 2$.

LEMMA 6.1. *Let $\mathcal{E}_{\mathbb{K}_k}$ be given for $k = 1, \ldots, t$. For all $a \in \mathbb{K}_t$ and $0 \leqslant i \leqslant \lg \kappa$, we can compute $a^{p^{2^i}}$ in time*

$$\bar{m}^{\omega-1}\kappa^{\epsilon(\kappa)}\tilde{O}(\log q).$$

**Proof.** For $k \leqslant t$, let $\mathsf{T}_k$ be the time needed to compute $b^{p^{2^i}}$ for any $b \in \mathbb{K}_k$ and $i \leqslant \lg \kappa$. Let

$$\sum_{j=0}^{m_t-1} a_j z_t^j \in \mathbb{K}_{t-1}[z_t]_{<m_t}$$

denote the canonical representative of $a \in \mathbb{K}_t$. We have

$$a^{p^{2^i}} = \sum_{j=0}^{m_t-1} a_j^{p^{2^i}} z_t^{jp^{2^i}} \bmod \mu_t(z_t) = \left(\sum_{j=0}^{m_t-1} a_j^{p^{2^i}} z_t^j\right) \circ \left(z_t^{p^{2^i}}\right) \bmod \mu_t(z_t).$$

Now $a_j^{p^{2^i}}$ can be computed recursively in time $\mathsf{T}_{t-1}$, so

$$\mathsf{T}_t = m_t \mathsf{T}_{t-1} + O(m_t^{\omega}\mathsf{M}_{\mathbb{K}_{t-1}}(1)).$$

From Proposition 6.1 it follows that

$$\mathsf{T}_t = m_t \mathsf{T}_{t-1} + m_t^{\omega-1}(m_1 \cdots m_{t-1})^{\epsilon(m_1\cdots m_{t-1})}\tilde{O}(m_1 \cdots m_t \log p).$$

We conclude by unrolling this identity. □

The auxiliary sequences $\mathcal{E}_{\mathbb{K}_k}$ are computed by induction using the following adaptation of Algorithm 2.2.

**Algorithm 6.1**

**Input.** $\mathbb{K}_k$, $\mathcal{E}_{\mathbb{K}_1}, \ldots, \mathcal{E}_{\mathbb{K}_{k-1}}$.

**Output.** $\mathcal{E}_{\mathbb{K}_k}$.

1. Compute $\mathcal{E}_0(z_k) := z_k^p \bmod \mu_k(z_k)$ using binary powering.
2. For $i = 1, \ldots, \lg \kappa$, write $\mathcal{E}_{i-1}(z_k) = \sum_{j=0}^{m_k-1} a_j z_k^j$ and compute $\mathcal{E}_i(z_k)$ as
$$\left(\sum_{j=0}^{m_k-1} a_j^{p^{2^{i-1}}} z_k^j\right) \circ \mathcal{E}_{i-1}(z_k) \bmod \mu_k(z_k).$$
3. Return $(\mathcal{E}_i(z_k))_{0 \leqslant i \leqslant \lg \kappa}$.

LEMMA 6.2. *Algorithm 2.1 is correct and runs in time*

$$(\bar{m}^{\omega-1}\log\kappa + \log p)\,\kappa^{\epsilon(\kappa)}\tilde{O}(m_1 \cdots m_k \log p).$$

**Proof.** We prove the correctness by induction on $i$. The case $i=0$ is clear. Assume that $i \geqslant 1$ and let $h(z_k) \in \mathbb{K}_{k-1}[z_k]$ be such that

$$\mathcal{E}_{i-1}(z_k) \;=\; \sum_{j=0}^{m_k-1} a_j z_k^j \;=\; z_k^{p^{2^{i-1}}} + h(z_k)\, \mu_k(z_k).$$

Then we have

$$\begin{aligned}
\left(\sum_{j=0}^{m_k-1} a_j^{p^{2^{i-1}}} z_k^j\right) \circ \mathcal{E}_{i-1}(z_k) \bmod \mu_k(z_k) \;&=\; \left(\sum_{j=0}^{m_k-1} a_j^{p^{2^{i-1}}} z_k^j\right) \circ \left(z_k^{p^{2^{i-1}}}\right) \bmod \mu_k(z_k) \\
&=\; \sum_{j=0}^{m_k-1} a_j^{p^{2^{i-1}}} z_k^{jp^{2^{i-1}}} \bmod \mu_k(z_k) \\
&=\; \left(\sum_{j=0}^{m_k-1} a_j z_k^j\right)^{p^{2^{i-1}}} \bmod \mu_k(z_k) \\
&=\; \left(z_k^{p^{2^{i-1}}}\right)^{p^{2^{i-1}}} \bmod \mu_k(z_k) \\
&=\; \mathcal{E}_i(z_k) \bmod \mu_k(z_k).
\end{aligned}$$

This completes the correctness proof. Concerning the complexity, the first step takes time $O(\mathsf{M}_{\mathbb{K}_{k-1}}(m_k) \log p)$, whereas the loop requires $\lg \kappa$ modular compositions and $m_k \lg \kappa$ computations of $p^{2^{i-1}}$-th powers in $\mathbb{K}_{k-1}$. By Lemma 6.1, the total running time is therefore bounded by

$$\begin{aligned}
&O((\mathsf{C}_{\mathbb{K}_{k-1}}(m_k) + m_k \bar{m}^{\omega-1} \kappa^{\epsilon(\kappa)} \tilde{O}(m_1 \cdots m_{k-1} \log p)) \log \kappa + \mathsf{M}_{\mathbb{K}_{k-1}}(m_k) \log p) \\
=\; &O((\mathsf{C}_{\mathbb{K}_{k-1}}(m_k) + \bar{m}^{\omega-1} \kappa^{\epsilon(\kappa)} \tilde{O}(m_1 \cdots m_k \log p)) \log \kappa + \mathsf{M}_{\mathbb{K}_{k-1}}(m_k) \log p).
\end{aligned}$$

Using the bounds

$$\begin{aligned}
\mathsf{M}_{\mathbb{K}_{k-1}}(m_k) \;&=\; \kappa^{\epsilon(\kappa)} \tilde{O}(m_1 \cdots m_k \log p) \\
\mathsf{C}_{\mathbb{K}_{k-1}}(m_k) \;&=\; m_k^{\omega-1} \kappa^{\epsilon(\kappa)} \tilde{O}(m_1 \cdots m_k \log p)
\end{aligned}$$

from Proposition 6.1, this yields the claimed cost. □

### 6.1.3. Irreducible factorization

COROLLARY 6.1. *Let $\mathbb{F}_q \equiv \mathbb{K}_t$ for a triangular tower as above. The computation of the irreducible factors of degree $\leqslant D$ of a polynomial of degree $d$ in $\mathbb{F}_q[x]$ takes an expected time*

$$(D^{0.5} d^{\omega-1} + \bar{m}^{\omega-1} + \log p)\, \kappa^{\epsilon(\kappa)}\, \tilde{O}(d \log q).$$

**Proof.** By Lemma 6.2, the auxiliary sequences $\mathcal{E}_{\mathbb{K}_1}, \ldots, \mathcal{E}_{\mathbb{K}_t}$ can be computed in time $(\bar{m}^{\omega-1} + \log p)\, \kappa^{\epsilon(\kappa)}\, \tilde{O}(\log q)$. By Proposition 6.1 and Lemma 6.1, we may take

$$\begin{aligned}
\Phi_{\mathbb{K}_t} \;&=\; \bar{m}^{\omega-1} \kappa^{\epsilon(\kappa)} \tilde{O}(\log q) \\
\mathsf{C}_{\mathbb{K}_t}(d) \;&=\; d^{\omega-1} \kappa^{\epsilon(\kappa)} \tilde{O}(d \log q).
\end{aligned}$$

The bound now follows from Theorem 4.1. □

COROLLARY 6.2. *Let $\mathbb{F}_q \equiv \mathbb{K}_t$ for a triangular tower as above. A polynomial of degree $d$ in $\mathbb{F}_q[x]$ can be tested to be irreducible in time*

$$(d^{\omega-1} + \bar{m}^{\omega-1} + \log p)\, \kappa^{\epsilon(\kappa)}\, \tilde{O}(d \log q).$$

**Proof.** This follows from Theorem 4.2, in a similar way as above. □

If $\bar{m} d \log p = O(\kappa^{o(1)})$, then we note that the bounds in Corollaries 6.1 and 6.2 further simplify into $(d \log q)^{1+o(1)}$, which has an optimal complexity exponent in terms of the input/output size.

## 6.2. Factoring over special primitive towers

In the case when the extension degree $\kappa$ of $\mathbb{F}_q$ over $\mathbb{F}_p$ is composite, we proposed various algorithms for modular composition [16] that are more efficient than the traditional "baby-step giant-step" method [30]. As before, these methods all represent $\mathbb{F}_q$ as the top field of a tower of finite fields

$$\mathbb{F}_p \equiv \mathbb{K}_0 \subset \mathbb{K}_1 \subset \cdots \subset \mathbb{K}_t \equiv \mathbb{F}_q. \tag{6.1}$$

Such towers can be built in several ways and each type of tower comes with its own specific complexity bounds for basic arithmetic operations and modular composition. In this section, we briefly recall the complexity bounds for the various types of towers and then combine them with the results of section 4.4.

The arithmetic operations in the fields $\mathbb{K}_i$ of the tower (6.1) are most efficient if each $\mathbb{K}_i$ is presented directly as a primitive extension $\mathbb{K}_i \equiv \mathbb{F}_p[y_i]/(\nu_i(y_i))$ over $\mathbb{F}_p$, where $\nu_i \in \mathbb{F}_p[y_i]$ is a monic polynomial of degree $m_1 \cdots m_i$. Towers of this type are called *primitive towers*. The primitive representations will be part of the precomputation. In [16], we studied the following types of primitive towers:

**Nested towers.** For $i = 2, \ldots, t$, we have $\nu_i = \nu_{i-1} \circ \tau_i$ , where $\tau_i \in \mathbb{F}_p[x]$ is a polynomial of degree $m_i$ (more generally, for a suitable generalization of composition, $\tau_i$ may even be a rational function of degree $m_i$); see [16, section 7.3] for details.

**Composed towers.** The $m_i$ are pairwise coprime and there exist monic irreducible polynomials $\lambda_1, \ldots, \lambda_t \in \mathbb{K}[z]$ of degrees $m_1, \ldots, m_t$ such that $\nu_1 = \lambda_1$ and $\nu_i = \nu_{i-1} \odot \lambda_i$ for $i = 2, \ldots, t$. Here $\odot$ stands for the composed product of irreducible polynomials; see [16, section 7.4].

**Artin–Schreier towers.** We have $m_1 = \cdots = m_t = p$ and the minimal polynomials of the successive extensions are given by

$$\begin{aligned} \mu_1(z_1) &= z_1^p - z_1 - 1 \\ \mu_2(z_2) &= z_2^p - z_2 - \alpha_1 && (i = 2, p = 2) \\ \mu_i(z_i) &= z_i^p - z_i - \alpha_{i-1}^{2p-1} && (\text{all other cases}), \end{aligned}$$

where $\alpha_i$ is a root of $\mu_i$ in $\mathbb{K}_i$ for $i = 1, \ldots, t$.

Composed towers and Artin–Schreier towers suffer from the inconvenience that they can only be used in specific cases. Nested towers are somewhat mysterious: many finite fields can be presented in this way, but we have no general proof of this empiric fact and no generally efficient way to compute such representations. From an asymptotic complexity point of view, nested towers are most efficient for the purposes of this paper, whenever they exist.

In [16], we have shown that one composition modulo $\nu_t$ can be done in time

$$\mathsf{C}_{\mathbb{F}_q/\mathbb{F}_p} = O(\mathsf{M}_{\mathbb{F}_p}(\bar{m}\kappa)\, \eta(\bar{m}, \kappa, t)),$$

where the overhead $\eta(\bar{m}, \kappa, t)$ depends as follows on the particular type of tower:

$$\eta(\bar{m}, \kappa, t) = \begin{cases} t \log \kappa & \text{nested towers} \\ \bar{m} \log \kappa & \text{composed towers} \\ \bar{m} \log^2 \kappa & \text{Artin–Schreier towers} \end{cases}$$

In the case of Artin–Schreier towers, we actually have $\mathsf{C}_{\mathbb{F}_q/\mathbb{F}_p} = O(p^2 \kappa \log^3 \kappa)$, which yields the announced value for $\eta(\bar{m}, \kappa, t)$ under the mild assumption that

$$\mathsf{M}_{\mathbb{F}_p}(n) = \Omega(n \log p \log(n \log p)).$$

By Lemma 2.3, the sequence $\mathcal{E}_{\mathbb{F}_q}$ can be computed in time

$$O(\mathsf{M}_{\mathbb{F}_p}(\bar{m}\kappa)\, \eta(\bar{m}, \kappa, t) \log \kappa + \mathsf{M}_{\mathbb{F}_p}(\kappa) \log p).$$

By Lemma 2.1, we may take

$$\Phi_{\mathbb{F}_q} = O(\mathsf{M}_{\mathbb{F}_p}(\bar{m}\kappa)\, \eta(\bar{m}, \kappa, t)).$$

Since we wish to avoid relying on the Kedlaya–Umans algorithm, we again use

$$\mathsf{C}_{\mathbb{F}_q}(d) = O(d^\omega \mathsf{M}_{\mathbb{F}_q}(1)).$$

Plugging these bounds into Theorems 4.1 and 4.2, we obtain:

COROLLARY 6.3. *Let $\mathbb{F}_q \equiv \mathbb{K}_t$ for a primitive tower of one of the above types. Then the computation of the irreducible factors of degree $\leqslant D$ of a polynomial of degree $d$ in $\mathbb{F}_q[x]$ can be done in expected time*

$$(D^{0.5} d^{\omega-1} + \bar{m}\, \eta(\bar{m}, \kappa, t) + \log p)\, \tilde{O}(d \log q).$$

COROLLARY 6.4. *Let $\mathbb{F}_q \equiv \mathbb{K}_t$ for a primitive tower of one of the above types. Then a polynomial of degree $d$ in $\mathbb{K}_t[x]$ can be tested to be irreducible in time*

$$(d^{\omega-1} + \bar{m}\, \eta(\bar{m}, \kappa, t) + \log p)\, \tilde{O}(d \log q).$$

In the particular case when $d\bar{m} \log p = (\log \kappa)^{O(1)}$, we note that both bounds further simplify into $\tilde{O}(d \log q)$, which is quasi-optimal.

## 7. CONCLUSION

We have revisited probabilistic complexity bounds for factoring univariate polynomials over finite fields and for testing their irreducibility. We mainly used existing techniques, but we were able to sharpen the existing bounds by taking into account recent advances on modular composition. However, the following major problems remain open:

- Do their exist practical algorithms for modular composition with a quasi-optimal complexity exponent?
- The existing bit complexity bounds for factorization display a quadratic dependency on the bit size $\log p$ of the prime field. Is this optimal?
- Is it possible to lower the complexity exponent 1.5 in $d$ for irreducible factorization? This problem is equivalent to several other ones, as explained in [12].

The improvements from this paper are most significant for finite fields of a large smooth extension degree over their prime field. Indeed, fast algorithms for modular composition were designed for this specific case in [16]. It would be interesting to know whether there are other special cases for which this is possible. Applications of such special cases would also be welcome.

## BIBLIOGRAPHY


**[1]** M. Ben-Or. Probabilistic algorithms in finite fields. In *22nd Annual Symposium on Foundations of Computer Science (SFCS 1981)*, pages 394–398. Los Alamitos, CA, USA, 1981. IEEE Computer Society.

**[2]** E. R. Berlekamp. Factoring polynomials over finite fields. *Bell System Technical Journal*, 46:1853–1859, 1967.

**[3]** E. R. Berlekamp. Factoring polynomials over large finite fields. *Math. Comput.*, 24:713–735, 1970.

**[4]** A. Bostan, F. Chyzak, M. Giusti, R. Lebreton, G. Lecerf, B. Salvy, and É. Schost. *Algorithmes Efficaces en Calcul Formel*. Frédéric Chyzak (self-published), Palaiseau, 2017. Electronic version available from `https://hal.archives-ouvertes.fr/AECF`.

**[5]** D. G. Cantor and H. Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Math. Comput.*, 36(154):587–592, 1981.

**[6]** Ph. Flajolet, X. Gourdon, and D. Panario. The complete analysis of a polynomial factorization algorithm over finite fields. *J. Algorithms*, 40(1):37–81, 2001.

**[7]** Ph. Flajolet and J.-M. Steyaert. A branching process arising in dynamic hashing, trie searching and polynomial factorization. In M. Nielsen and E. Schmidt, editors, *Automata, Languages and Programming*, volume 140 of *Lect. Notes Comput. Sci.*, pages 239–251. Springer–Verlag, 1982.

**[8]** J. von zur Gathen. Who was who in polynomial factorization. In *ISSAC'06: International Symposium on Symbolic and Algebraic Computation*, pages 1–2. New York, NY, USA, 2006. ACM.

**[9]** J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, New York, 3rd edition, 2013.

**[10]** J. von zur Gathen and G. Seroussi. Boolean circuits versus arithmetic circuits. *Inf. Comput.*, 91(1):142–154, 1991.

**[11]** J. von zur Gathen and V. Shoup. Computing Frobenius maps and factoring polynomials. *Comput. complexity*, 2(3):187–224, 1992.

**[12]** Z. Guo, A. K. Narayanan, and Ch. Umans. Algebraic problems equivalent to beating exponent 3/2 for polynomial factorization over finite fields. `https://arxiv.org/abs/1606.04592`, 2016.

**[13]** D. Harvey and J. van der Hoeven. Faster polynomial multiplication over finite fields using cyclotomic coefficient rings. *J. Complexity*, 54:101404, 2019.

**[14]** D. Harvey and J. van der Hoeven. Polynomial multiplication over finite fields in time $O(n \log n)$. Technical Report, HAL, 2019. `http://hal.archives-ouvertes.fr/hal-02070816`.

**[15]** J. van der Hoeven. *The Jolly Writer. Your Guide to GNU TeXmacs*. Scypress, 2020.

**[16]** J. van der Hoeven and G. Lecerf. Modular composition via factorization. *J. Complexity*, 48:36–68, 2018.

**[17]** J. van der Hoeven and G. Lecerf. Accelerated tower arithmetic. *J. Complexity*, 55:101402, 2019.

**[18]** J. van der Hoeven and G. Lecerf. Fast amortized multi-point evaluation. Technical Report, HAL, 2020. `https://hal.archives-ouvertes.fr/hal-02508529`.

**[19]** J. van der Hoeven and G. Lecerf. Fast multivariate multi-point evaluation revisited. *J. Complexity*, 56:101405, 2020.

**[20]** X. Huang and V. Y. Pan. Fast rectangular matrix multiplication and applications. *J. Complexity*, 14(2):257–299, 1998.

**[21]** E. Kaltofen. Polynomial factorization: a success story. In *ISSAC '03: Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation*, pages 3–4. New York, NY, USA, 2003. ACM.

**[22]** E. Kaltofen and V. Shoup. Subquadratic-time factoring of polynomials over finite fields. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '95, pages 398–406. New York, NY, USA, 1995. ACM.

**[23]** E. Kaltofen and V. Shoup. Fast polynomial factorization over high algebraic extensions of finite fields. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*, ISSAC '97, pages 184–188. New York, NY, USA, 1997. ACM.

**[24]** E. Kaltofen and V. Shoup. Subquadratic-time factoring of polynomials over finite fields. *Math. Comput.*, 67(223):1179–1197, 1998.

**[25]** K. S. Kedlaya and C. Umans. Fast modular composition in any characteristic. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2008)*, pages 146–155. Los Alamitos, CA, USA, 2008. IEEE Computer Society.

**[26]** K. S. Kedlaya and C. Umans. Fast polynomial factorization and modular composition. *SIAM J. Comput.*, 40(6):1767–1802, 2011.

**[27]** G. Lecerf. Fast separable factorization and applications. *Appl. Algebra Engrg. Comm. Comput.*, 19(2):135–160, 2008.

**[28]** G. L. Mullen and D. Panario. *Handbook of Finite Fields*. Chapman and Hall/CRC, 2013.

**[29]** V. Neiger, J. Rosenkilde, and G. Solomatov. Generic bivariate multi-point evaluation, interpolation and modular composition with precomputation. In A. Mantzaflaris, editor, *Proceedings of the 45th International Symposium on Symbolic and Algebraic Computation*, ISSAC '20, pages 388–395. New York, NY, USA, 2020. ACM.

**[30]** M. S. Paterson and L. J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.*, 2(1):60–66, 1973.

**[31]** A. Poteaux and É. Schost. Modular composition modulo triangular sets and applications. *Comput. Complex.*, 22(3):463–516, 2013.

**[32]** M. O. Rabin. Probabilistic algorithms in finite fields. *SIAM J. Comput.*, 9(2):273–280, 1980.

**[33]** V. Shoup. Fast construction of irreducible polynomials over finite fields. *J. Symbolic Comput.*, 17(5):371–391, 1994.

**[34]** V. Shoup. A new polynomial factorization algorithm and its implementation. *J. Symbolic Comput.*, 20(4):363–397, 1995.

**[35]** V. Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2nd edition, 2008.